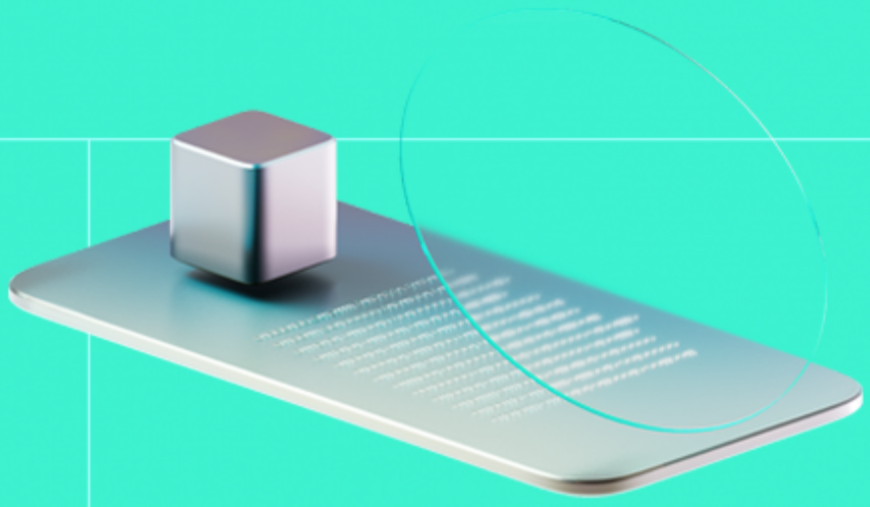
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Sky Option

Date: 19/08/2025

We express our gratitude to the Sky Option team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

SkyCoin (XSO) is an enhanced ERC20 token that integrates burn and pause capabilities, anti-whale protections, blacklist controls, and owner-configurable limits to provide a secure, compliant, and adaptable digital asset.

Document

Name	Smart Contract Code Review and Security Analysis Report for Sky Option
Audited By	Panagiotis Konstantinidis, Franco Bregante
Approved By	Ivan Bondar
Website	https://www.skyxso.com
Changelog	11/08/2025 - Preliminary Report
	13/08/2025 - Remediation Report
	19/08/2025 - Final Report
Platform	BSC
Language	Solidity
Tags	Fungible Token, ERC-20
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/skyoption/xso
Initial Commit	97ac21a
Remediation Commit	c49d600
Deployed Address	https://bscscan.com/token/0xfb40a811FB2568de9709476c09935a6A0DAFA6aa

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

5	5	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	1

Vulnerability	Severity
F-2025-12072 - Improper Wallet Limit Check in _checkLimits for DEX Pair Address Enables Anti-Whale Restriction Bypass or Blocks Token Sales	Medium
F-2025-12067 - Misleading Documentation in emergencyRemoveLimits Allows Limits to Be Reinstated	Low
F-2025-12065 - Floating Pragma	Info
F-2025-12066 - Unused ReentrancyGuard Increases Code Complexity Without Providing Protection	Info
F-2025-12068 - Inefficient Code Patterns and Unused Errors Reduce Code Efficiency	Info

Documentation quality

- Functional and technical requirements are clearly outlined.
- Security measures like access control, and input validation are well-documented.
- Detailed deployment, testing, and verification procedures are provided.
- Documentation is clear and whitepaper is provided.

Code quality

- NatSpec comments are provided.
- The development environment is configured.
- Organized with separate sections for functionality.

Test coverage

Code coverage of the project is **92%** (branch coverage).

- Core functionality and edge cases are covered.
- Negative scenarios, such as invalid inputs and unauthorized access, are tested.

Table of Contents

System Overview **6**

Privileged Roles 6

Potential Risks **7**

Findings **8**

Vulnerability Details 8

Disclaimers 19

Appendix 1. Definitions **20**

Severities 20

Potential Risks 20

Appendix 2. Scope **21**

Appendix 3. Additional Valuables **22**

System Overview

SkyCoin (XSO) is an ERC20-based token contract that incorporates burn and pause features, anti-whale protections, blacklist controls, and owner-configurable limits to enhance security and compliance. It is built on top of OpenZeppelin's ERC20, ERC20Burnable, ERC20Pausable, and Ownable modules.

The token implements the following functionalities:

- **Burn**: Allows holders to destroy their tokens, reducing the total supply.
- **Pause/Unpause**: Enables the owner to halt or resume functionalities of the token in emergency situations.
- **Blacklist**: Blocks blacklisted addresses from transferring tokens.
- **Anti-Whale Protections**: Enforces configurable maximum transaction amounts and wallet balances.
- **Exemption List**: Allows specified addresses to bypass anti-whale restrictions.
- **Configurable Limits**: The owner can update transaction and wallet limits or disable them entirely.

It has the following attributes:

- Name: Sky Coin
- Symbol: XSO
- Decimals: 18
- Total Supply: 1,000,000,000,000 tokens (1 trillion XSO) minted at deployment to the specified recipient address.

Privileged roles

The owner of the SkyCoin contract holds full administrative control over all system parameters and restrictions. Specifically, the owner can:

- Pause and unpause all token transfers at any time.
- Blacklist or unblacklist any address, preventing them from sending or receiving tokens.
- Update maximum transaction amounts and wallet balances, directly altering anti-whale protections.
- Toggle the anti-whale limits on or off, or remove them entirely through the emergency function.
- Grant or revoke exemption status for any address, allowing it to bypass anti-whale restrictions.

This level of control allows the owner to arbitrarily modify the operational rules governing token transfers and restrictions. As a result, the owner has the ability to significantly alter the token's transfer behavior, disable protections, or restrict participation for any address. At present, the owner of the contract is a multi-signature wallet, which reduces the risk of a single point of failure or unilateral misuse of these privileges.

Potential Risks

- **Centralized Minting to a Single Address**: The project concentrates minting tokens in a single address, raising the risk of fund mismanagement or theft, especially if key storage security is compromised.
- **Owner's Unrestricted State Modification:** The owner can arbitrarily modify transaction limits, wallet limits, blacklist status, and exemption lists, enabling changes to anti-whale protections and transfer behavior at any time. Such unrestricted control introduces centralization risk and may affect contract integrity and user trust if misused or compromised. These privileges could also be configured in a way that prevents token holders from transferring or selling their tokens, effectively locking them or making them unsellable.
- **Absence of Time-lock Mechanisms for Critical Operations:** The contract does not implement any delay or review mechanism for sensitive administrative actions. This allows immediate execution of potentially impactful changes, such as disabling limits or blacklisting addresses, without providing stakeholders an opportunity to review or react, increasing the risk of misuse.
- **Administrative Key Control Risks:** All sensitive configuration changes, including blacklisting, limit adjustments, and pausing transfers, are gated by a single administrative key. If this key is compromised, the attacker could immediately alter transfer conditions, disable protections, and impact token holder activity.

Findings

Vulnerability Details

F-2025-12072 - Improper Wallet Limit Check in _checkLimits for DEX Pair Address Enables Anti-Whale Restriction Bypass or Blocks Token Sales - Medium

Description:

The contract enforces anti-whale restrictions via the internal `_checkLimits()` function, which is triggered on every token transfer (except mint and burn) through the `_update()` override:

```
function _update(address from, address to, uint256 value)
    internal
    override(ERC20, ERC20Pausable)
{
    ...

    if (limitsEnabled && from != address(0) && to != address(0)) {
        _checkLimits(from, to, value);
    }

    super._update(from, to, value);
}
```

This logic ensures that each transfer does not exceed `maxTransactionAmount` and that the recipient's balance does not surpass `maxWalletBalance`. These checks are intended to prevent large accumulations of tokens by a single address. However, `_checkLimits()` does not explicitly exclude the DEX liquidity pair address from the `maxWalletBalance` check:

```
    function _checkLimits(address from, address to, uint256 value) internal v
iew {
        // Skip limits for exempt addresses
        if (exemptFromLimits[from] || exemptFromLimits[to]) {
            return;
        }

        // Check transaction limit
        if (value > maxTransactionAmount) {
            revert ExceedsMaxTransaction();
```

```
        }

        // Check wallet limit for recipient
        if (balanceOf(to) + value > maxWalletBalance) {
            revert ExceedsMaxWallet();
        }
    }
```

In a standard deployment, the pair address must be exempted from limits to avoid blocking sales once its balance exceeds `maxWalletBalance`. However, exempting the pair also causes the `return` statement to trigger, skipping **all** checks in `_checkLimits()` for both buy and sell transactions. This means a user can bypass not only wallet balance limits but also per-transaction limits when interacting through the pair.

If the pair is not exempted, sales will eventually be blocked once its balance surpasses the wallet limit. The pair address, which facilitates trading, is expected to hold a large portion of the token supply, particularly during liquidity provision or after multiple purchases, making it impractical to subject it to standard wallet balance restrictions.

Depending on configuration, this design either blocks sales through the pair once its balance grows beyond `maxWalletBalance` or fully bypasses all anti-whale checks for trades through the pair, including buys. This undermines the intended enforcement of trading restrictions and can negatively affect trading behaviour and liquidity management.

Assets:

- xso-contract.sol [https://github.com/skyoption/xso]

Status: Fixed

Classification

Impact Rate: 4/5

Likelihood Rate: 5/5

Exploitability: Dependent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: Update the `_checkLimits()` logic to exclude the pair address from wallet balance enforcement. This ensures that trading via the DEX remains functional while still enforcing wallet limits for user accounts.

Example Fix:

```
address public pairAddress;
bool private pairSet;

function setPairAddress(address _pair) external onlyOwner {
    require(!pairSet, "Pair already set");
    require(_pair != address(0), "Zero address");
    pairAddress = _pair;
    pairSet = true;
}

function _checkLimits(address from, address to, uint256 value) internal view
{
    if (exemptFromLimits[from] || exemptFromLimits[to]) return;

    if (value > maxTransactionAmount) revert ExceedsMaxTransaction();

    // Skip wallet limit check for the pair address
    if (to != pairAddress && balanceOf(to) + value > maxWalletBalance) {
        revert ExceedsMaxWallet();
    }
}
```

This modification preserves anti-whale protections while ensuring the trading mechanism remains unrestricted and functional.

Resolution: **Fixed** at commit `8cb793b`. A `pairAddress` mapping was added with a one-time setter, and `_checkLimits()` function was updated to skip only the wallet limit for the pair while keeping transaction limits.

F-2025-12067 - Misleading Documentation in emergencyRemoveLimits Allows Limits to Be Reinstated - Low

Description:

The contract includes the `emergencyRemoveLimits()` function, which is documented as an irreversible emergency mechanism to permanently remove transaction and wallet limits, according to the NatSpec comments provided in the code:

```
/**
 * @dev Emergency function to remove limits (use with caution)
 * @notice This permanently disables all limits and cannot be undone
 */
function emergencyRemoveLimits() external onlyOwner {
    limitsEnabled = false;
    maxTransactionAmount = totalSupply();
    maxWalletBalance = totalSupply();
    emit LimitsToggled(false);
    emit LimitsUpdated(totalSupply(), totalSupply());
}
```

However, despite the claim that “This permanently disables all limits and cannot be undone” the contract still allows the owner to call the `updateLimits()` and `toggleLimits()` functions even after `emergencyRemoveLimits()` has been executed.

```
function updateLimits(uint256 _maxTransaction, uint256 _maxWallet) external o
nlyOwner { ... }

function toggleLimits(bool enabled) external onlyOwner { ... }
```

These two functions allow the owner to reactivate the limits or change them arbitrarily at any point after `emergencyRemoveLimits()` is called. As such, the “permanent” effect described in the comment is misleading, and the actual behavior does not enforce any form of irreversibility.

This discrepancy could result in confusion for integrators, or users, who may rely on the documentation and assume the limits can no longer be reinstated once disabled.

Assets:

- xso-contract.sol [https://github.com/skyoption/xso]

Status: Fixed

Classification

Impact Rate: 2/5

Likelihood Rate: 3/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: To resolve this inconsistency, either update the comment of the `emergencyRemoveLimits()` function to accurately reflect that the action is reversible, or modify the function's implementation to make the removal of limits truly permanent, for example by introducing an immutable flag that prevents any future updates to the limit values or enforcement.

Resolution: **Fixed** at commit `8018d26`. The NatSpec comment for `emergencyRemoveLimits()` function was updated to state that the action is reversible, matching the actual contract behavior.

F-2025-12065 - Floating Pragma - Info

Description: In Solidity development, the pragma directive specifies the compiler version to be used, ensuring consistent compilation and reducing the risk of issues caused by version changes. However, using a floating pragma (e.g., `^0.8.20`) introduces uncertainty, as it allows the contract to be compiled with any version within the specified range. This can result in discrepancies between the compiler used in testing and the one used during deployment, increasing the likelihood of vulnerabilities or unexpected behavior due to changes in compiler versions.

The project currently uses floating pragma declarations (`^0.8.20`) in its Solidity contracts. This increases the risk of deploying with a compiler version different from the one tested, potentially reintroducing known bugs from older versions or causing unexpected behavior with newer versions. These inconsistencies could result in security vulnerabilities, system instability, or financial loss. Locking the pragma version to a specific, tested version is essential to prevent these risks and ensure consistent contract behavior.

```
pragma solidity ^0.8.20;
```

Assets:

- xso-contract.sol [https://github.com/skyoption/xso]

Status: Fixed

Classification

Impact Rate: 2/5

Likelihood Rate: 1/5

Exploitability: Dependent

Complexity: Simple

Severity: Info

Recommendations

Remediation: It is recommended to lock the pragma version to the specific version that was used during development and testing. This ensures that the contract will always be compiled with a known, stable compiler

version, preventing unexpected changes in behavior due to compiler updates. For example, instead of using `^0.8.xx`, explicitly define the version with `pragma solidity 0.8.20;`.

Before selecting a version, review known bugs and vulnerabilities associated with each Solidity compiler release. This can be done by referencing the official Solidity compiler release notes: Solidity GitHub releases or Solidity Bugs by Version. Choose a compiler version with a good track record for stability and security.

Resolution: **Fixed** at commit `b71ec98`. The floating pragma was replaced with the fixed version `0.8.30` to ensure consistent compilation and prevent version-related issues.

F-2025-12066 - Unused ReentrancyGuard Increases Code Complexity Without Providing Protection - Info

Description:

The contract inherits from OpenZeppelin's `ReentrancyGuard`:

```
import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
...
contract SkyCoin is ERC20, ERC20Burnable, ERC20Pausable, Ownable, ReentrancyGuard {
```

By doing so, the contract has access to the `nonReentrant` modifier, which is commonly used to prevent reentrancy attacks in state-changing functions that involve external calls. However, throughout the implementation, the `nonReentrant` modifier is never applied to any function. This indicates that the reentrancy guard is not actively used to protect any execution flow.

As a result, the inclusion of `ReentrancyGuard` increases the contract's size and perceived security guarantees without actually enforcing any reentrancy protection. This may introduce confusion to those who assume reentrancy protection is present when it is not. Moreover, its unused presence may suggest a missed opportunity to secure functions that perform external interactions, especially if such functions are later added or modified.

Assets:

- xso-contract.sol [https://github.com/skyoption/xso]

Status: Fixed

Classification

Impact Rate: 2/5

Likelihood Rate: 1/5

Exploitability: Independent

Complexity: Simple

Severity: Info

Recommendations

Remediation: Since the `ReentrancyGuard` is not used anywhere in the current contract, remove the import and inheritance of `ReentrancyGuard` to reduce unnecessary code complexity and avoid misleading assumptions about reentrancy protection. If in the future a function performs external calls and requires reentrancy protection, the modifier can be reintroduced explicitly where needed.

Resolution: **Fixed** at commit `c49d600`. The unused `ReentrancyGuard` inheritance was removed to reduce contract size and avoid misleading security assumptions.

F-2025-12068 - Inefficient Code Patterns and Unused Errors Reduce Code Efficiency - Info

Description: The contract contains several instances of redundant or unoptimized code that reduce overall code clarity and maintainability, specifically:

- **Unused Custom Error:**
  The custom error `LimitsNotEnabled()` is declared but never used throughout the contract:

```
error LimitsNotEnabled();
```

- **Redundant Constant Assignment:**
  The constant `MAX_SUPPLY` is declared and then immediately reassigned to `INITIAL_SUPPLY`:

```
uint256 public constant MAX_SUPPLY = 1_000_000_000_000 * 10**18;
uint256 public constant INITIAL_SUPPLY = MAX_SUPPLY;
```

This introduces unnecessary indirection and duplication, as `INITIAL_SUPPLY` is simply repeating the value of `MAX_SUPPLY` without modification.

- **Unnecessary Calculations:**
  In the constructor, the `maxWalletBalance` is set with a redundant multiplication and division:

```
maxWalletBalance = INITIAL_SUPPLY * 100 / 100;
```

This is equivalent to `maxWalletBalance = INITIAL_SUPPLY;` and the intermediate operations offer no functional benefit.

- **Unclear Exemption Assignment:**
  The `exemptFromLimits[address(this)] = true;` assignment lacks a clear functional purpose. The contract is not designed to participate in token transfers, holds no logic for sending tokens from its own balance, and there is no apparent scenario where transferring tokens into the contract would be necessary.

```
exemptFromLimits[address(this)] = true;
```

As a result, exempting the contract's own address from transfer limits introduces unnecessary code without providing any practical

benefit.

Although these issues do not pose any security risk, they reduce the overall clarity of the code and may create distractions during reviews or future modifications. Maintaining clean and optimized logic is particularly important for smart contracts, where auditability and minimalism are crucial.

Assets:

- xso-contract.sol [https://github.com/skyoption/xso]

Status: Fixed

Classification

Impact Rate: 1/5

Likelihood Rate: 2/5

Exploitability: Independent

Complexity: Simple

Severity: Info

Recommendations

Remediation: To improve the clarity, efficiency, and maintainability of the codebase, the following optimizations are recommended:

- Remove the unused custom error `LimitsNotEnabled()` to reduce dead code and avoid misleading future developers about non-existent logic paths.
- Eliminate the redundant `INITIAL_SUPPLY` constant and directly use `MAX_SUPPLY` where applicable, since both represent the same value.
- Simplify expressions like `* 100 / 100` by directly assigning the intended value (e.g., `= MAX_SUPPLY`) to avoid unnecessary operations and improve readability.
- Remove the `exemptFromLimits[address(this)] = true;` assignment, as the contract is not designed to transfer tokens and there is no practical reason for it to be exempt from transfer limits.

Resolution: **Fixed** at commit `d39a7f7`. Redundant code and unused elements were removed, and calculations were simplified to improve clarity and maintainability.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

As part of Hacken’s ongoing quality assurance process, we may conduct re-audits of select projects. These re-audits are performed independently from the original audit and are intended solely for internal quality control and improvement. Updated reports resulting from such re-audits will be shared privately with the respective clients and may be published on the Hacken website only with their explicit consent.
The sole authoritative source for finalized and most up-to-date versions of all reports remains the Audits section at https://hacken.io/audits/.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/skyoption/xso
Initial Commit	97ac21ad3c0b5630977f6bee3111e26c68613d94
Remediation Commit	c49d6001c358b3855453c2cc80c1b23e2945a510
Deployed Address	https://bscscan.com/token/0xfb40a811FB2568de9709476c09935a6A0DAFA6aa
Whitepaper	Shared internally
Requirements	README.md
Technical Requirements	TECHNICAL_DOCUMENTATION.md

Asset	Type
xso-contract.sol [https://github.com/skyoption/xso]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.